```
# −*− coding: utf−8 −*−
"""
Created on Sat Jul  2 15:36:57 2016

@author: guy barat
"""

"""
X − PSI − PT, 2016
"""
# PARTIE I

# Question 1
def deplacerParticule(particule, largeur, hauteur):
    x, y, vx, vy = particule
    X, Y, VX, VY = x + vx, y + vy, vx, vy
    if not 0 <= X <= largeur:
        X −= 2*vx
        VX *= −1
    if not 0 <= Y <= hauteur:
        Y = y − vy
        VY = −VY
    return X, Y, VX, VY

def essai_qu1():
    largeur, hauteur = 6, 4
    P = [(2.2, 1.6, 0.1, 0.6), (3.25, 2.45, 0.45, 0.05), (0.2, 2.7, −0.5, −0.3),
\
        (5.7, 3.7, 0.5, 0.6)]
    return [deplacerParticule(x, largeur, hauteur) for x in P]

# Partie II

# Question 2
def nouvelleGrille(largeur, hauteur):
    return[[None for i in range(hauteur)] for j in range(largeur)]


# Question 3
def majGrilleOuCollision(grille):
    largeur, hauteur = len(grille), len(grille[0])
    nouv_grille = nouvelleGrille(largeur, hauteur)
    positions = []
    for i in range(largeur):
        for j in range(hauteur):
            if grille[i][j] is not None:
                new = deplacerParticule(grille[i][j], largeur, hauteur)
                a, b = int(new[0]), int(new[1])
                if (a, b) in positions:
                    return None
                nouv_grille[a][b] = new
                positions.append((a, b))
    return nouv_grille

gri = [[None, None, (0.2, 2.7, −0.5, −0.3), None],[None, None, None, None],\
    [None, (2.2, 1.6, 0.1, 0.6), None, None],\
    [None, None, (3.25, 2.45, 0.45, 0.05) , None],\
    [None, None, None, None],[None, None, None, (5.7, 3.7, 0.5, 0.6)]]

def essai_qu3():
    U = majGrilleOuCollision(gri)
    V = majGrilleOuCollision(U)
```



```
    return U, V


# Question 4
def attendreCollisionGrille(grille, tMax):
    t = 0
    while t < tMax and grille is not None:
        grille = majGrilleOuCollision(grille)
        t += 1
    if grille is None:
        return t
    return None

def essai_qu4():
    return attendreCollisionGrille(gri, 1), attendreCollisionGrille(gri, 4)


# Question 5
"""
− Complexité de deplacerParticule : O(1) ;
− complexité de majGrilleOuCollision : O(largeur*hauteur) ;
− complexité de attendreCollisionGrille : O(largeur*hauteur*tMax).
"""


# PARTIE III

# Question 6
def detecterCollisionEntreParticules(p1, p2):
    dist2 = (p1[0] − p2[0])**2 + (p1[1] − p2[1])**2
    return dist2 <= 4*rayon**2


# Question 7
def maj(particules):
    largeur, hauteur = particules[0], particules[1]
    return (largeur, hauteur, \
        [deplacerParticule(P, largeur, hauteur) for P in particules[2]])


# Question 8
def majOuCollision(particules):
    Dep = maj(particules)[2]
    for i in range(1, len(Dep)):
        for j in range(i):
            if detecterCollisionEntreParticules(Dep[i], Dep[j]):
                return None
    return (particules[0], particules[1], Dep)


# Question 9
def attendreCollision(particules, tMax):
    t = 0
    while t < tMax and particules is not None:
        particules = majOuCollision(particules)
        t += 1
    if particules is None:
        return t
    return None

part = (6, 4, [(2.2, 1.6, 0.1, 0.6), (3.25, 2.45, 0.45, 0.05),\
     (0.2, 2.7, −0.5, −0.3), (5.7, 3.7, 0.5, 0.6)])
```

```
rayon = 1

def essai_qu6a9(tMax):
    U = majOuCollision(part)
    V = majOuCollision(U)
    print(U, V)
    return attendreCollision(part, tMax)

"""
Pour n particules et tmax :
− complexité de detecterCollisionEntreParticules : O(1) ;
− complexité de maj : O(n) ;
− complexité de majOuCollision : O(n) + O(n**2) = O(n**2) ;
− complexité de attendreCollision : O(tMax * n**2).
"""

# Question 10
"""
Pour que deux particules puissent entrer en collision à l'instant suivant,
il faut qu'elle se trouvent à une distance inférieure ou égale à
2*(vMax + rayon) − ce qui correspond à des vitesses opposées de norme vMax.
"""

# Question 11
def majOuCollisionX(particules):
    Dep = maj(particules)[2]
    n = len(particules[2])
    for i in range(n − 1):
        j = i + 1
        while j < n and particules[2][j][0] <= particules[2][i][0] + 2*rayon:
            if detecterCollisionEntreParticules(particules[2][i], particules[2][
j]):
                return None
            j += 1
    return (particules[0], particules[1], Dep)

def essai_qu11():
    return majOuCollisionX(part)


# PARTIE IV
# Question 12
def scm(s):
    L, i, n = [], 0, len(s)
    while i < n:
        j = i + 1
        while j < n and s[j] >= s[j−1]:
            j += 1
        L.append((i, j − 1))
        i = j
    return L

# Question 13

def fusionner(s, r1, r2):
    d1, f1 = r1
    d2, f2 = r2
    L = []
    while d1 <= f1 and d2 <= f2:
        if s[d1] <= s[d2]:
            L.append(s[d1])
            d1 += 1
```



```
        else:
            L.append(s[d2])
            d2 += 1
    if d1 > f1:
        for x in s[d2: f2+1]:
            L.append(x)
    else:
        for x in s[d1: f1+1]:
            L.append(x)
    for i in range(r2[1] − r1[0] + 1):
        s[r1[0] + i] = L[i]


# Question 14
def depileFusionneRemplace(s, pile):
    r2 = pile.pop()
    r1 = pile.pop()
    fusionner(s, r1, r2)
    pile.append((r1[0], r2[1]))


# Question 15
def alphaTri(s):
    def longueur(segment):
        return segment[1] − segment[0] + 1
    liste_scm = scm(s)
    nb_scm = len(liste_scm)
    pile, h, i = [liste_scm[0]], 1, 1
    # h est la hauteur de la pile, i l'indice de la dernière scm traitée
    while i < nb_scm: # première phase
        if h >= 2:
            top = pile.pop()
            sous_top = pile.pop()
            pile.append(sous_top)
            pile.append(top)
            if longueur(sous_top) < 2*longueur(top):
                depileFusionneRemplace(s, pile)
                h −= 1
            else:
                pile.append(liste_scm[i])
                i += 1
                h += 1
        else:
            pile.append(liste_scm[i])
            i += 1
            h += 1
    while h > 1: # deuxième phase
        depileFusionneRemplace(s, pile)
        h −= 1

"""
La structure de pile, qui ne possède ni de méthode len, ni de lecture d'éléments
ainsi que l'usage de la fonction depileFusionneRemplace entraînent un programme
assez maladroit : deux else identiques, dépilements pour lecture suivis
de réempilements avant de recommencer par appel de depileFusionneRemplace...
"""

def essai_qu12a15():
    s = [3, 4, 8, 11, 1, 5, 2, 7, 9, 0, 10, 0]
    scms = scm(s)
    alphaTri(s)
    print("décomposition en scm : {}\n alphaTri : {}".format(scms, s))
```